ELSEVIER

Robotics and Autonomous Systems 37 (2001) 115–125

Robotics and Autonomous Systems

www.elsevier.com/locate/robot

Real-time visual system for interaction with a humanoid robot

Aleš Ude*, Tomohiro Shibata [c], Christopher G. Atkeson [a,d]

[a] *ATR-I, Information Sciences Division, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*
[b] *Department of Automatics, Biocybernetics and Robotics, Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia*
[c] *Japan Science and Technology Corporation, ERATO Kawato Dynamic Brain Project, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*
[d] *Carnegie Mellon University, Robotics Institute, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA*

Abstract

In this paper, we describe a new real-time visual system that enables a humanoid robot to learn from and interact with humans. The core of the visual system is a probabilistic tracker that uses shape and color information to find relevant objects in the scene. Multiscale representations, windowing and masking are employed to accelerate the data processing. The perception system is directly coupled with the motor control system of our humanoid robot DB. We present two case studies of on-line interaction with a humanoid robot: mimicking of human hand motion and smooth pursuit of human head motion. The generation of humanoid robot motion based on the position of relevant body parts is accomplished in real time. Both studies are supported by experimental results on DB. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Real-time visual tracking; Humanoid robots; Mimicking; Smooth pursuit

1. Introduction

We are currently investigating ways to program and interact with a humanoid robot. Movement imitation or mimicking and higher forms of learning from demonstrations have been identified as a useful tool for programming such robots [1,11]. To learn from demonstrations and to interact with humans, the humanoid must be able to perceive human motion. While off-line processing of visual data is sometimes acceptable for learning from demonstration [14,15], a real-time perceptual system is essential for interaction tasks. Once motion perception is seen as a continuous process that interacts with the motor system, the required standards of reliability become much more stringent because failure in just one image frame might cause the entire system to break down.

Our humanoid robot DB (see Figs. 3 and 5) has 30 degrees of freedom: seven for each arm, three for each leg, two for each eye, three for the head and three for the torso. Each eye of the robot's oculomotor system consists of two cameras, a wide-angle (100 degrees view angle horizontally) color camera for peripheral vision, and a second narrow view camera (24 degrees view angle horizontally) providing a color image for foveal vision. This setup mimics the foveated retinal structure of primates. Such setup is essential for an artificial vision system in order to obtain high resolution images of objects of interest while still being able to perceive events in the peripheral environment. The images from the wide-angle cameras are captured and processed by standard PCs running the Windows NT operating system. The extracted data is sent via serial connections to a Power PC processor that generates data needed by a motor control system.

The key issue when realizing a real-time motion perception system is to avoid excessive interaction

* Corresponding author.
E-mail address: ude@isd.atr.co.jp (A. Ude).

PII: S0921-8890(01)00153-1

between the pieces of data in both the time and spatial domains. A practical system for perceiving human motion should be able to deal with complex environments and at least moderately changing lighting conditions. Probabilistic approaches are the prime candidates to explore because they allow us to prevent excessive data interaction through independency assumptions and because continuous probabilities associated with image pixels prevent the perceptual algorithms from being brittle with respect to the variations in the background and lighting conditions. By putting the motion tracking and estimation problems in a Bayesian setting, we can utilize a maximum likelihood approach to find the relevant objects and to recover the observed motion. This should enable reliable motion tracking once the real-time process is allowed to run.

2. Motion perception system

The goal of our real-time visual system is to perceive motion of body parts such as hands and head as well as objects that the observed person is manipulating. The most important part of the system is a real-time tracker, which we present in this section. We also consider some important issues such as 2D shape estimation, occlusion reasoning, and 3D estimation of positions of the tracked entities using stereo.

2.1. Probabilistic framework

We represent the observed environment by a number of random processes. Each entity to be tracked is represented by one process. Let us denote the probability that a pixel positioned at $u = (u, v)$ having color intensity I_u was generated by the process Θ_k, $k = 1, \ldots, K$, by $P(I_u, u|\Theta_k)$. We also introduce two additional processes: the optional background process Θ_{K+1}, which describes the stationary background (useful only for fixed cameras), and the outlier process Θ_0, which models the data not captured by other processes. Assuming that every pixel stems from one of the mutually independent processes Θ_k, $k = 0, \ldots, K+1$ (closed-world assumption), we can write the probability that color I_u was observed at location u using the total probability law

$$P(I_u, u|\boldsymbol{\Theta}) = \sum_{k=0}^{K+1} \omega_k P(I_u, u|\Theta_k), \tag{1}$$

where ω_k is the prior (mixture) probability to observe the process Θ_k, $\sum_{k=0}^{K+1} \omega_k = 1$, and $\boldsymbol{\Theta} = \{\Theta_0, \Theta_1, \ldots, \Theta_{K+1}\}$. Finally, neglecting the correlation of assigning neighboring pixels to processes, we can evaluate the overall probability to observe the image $\boldsymbol{I}$:

$$P(\boldsymbol{I}) = P(\boldsymbol{I}|\boldsymbol{\Theta}) = \prod_u P(I_u, u|\boldsymbol{\Theta}). \tag{2}$$

At each time step, we would like to determine $(\Theta_1, \ldots, \Theta_K, \omega_0, \omega_1, \ldots, \omega_{K+1})$ so that the likelihood (2) is maximized. Instead of maximizing (2), it is often easier to minimize the negative log likelihood

$$\begin{aligned} L(\boldsymbol{\Theta}, \boldsymbol{\omega}) &= -\log(P(\boldsymbol{I}|\boldsymbol{\Theta})) \\ &= -\sum_u \log(P(I_u, u|\boldsymbol{\Theta})), \end{aligned} \tag{3}$$

where $\boldsymbol{\omega} = (\omega_0, \ldots, \omega_{K+1})$. Taking into account that mixture probabilities should add up to 1, the corresponding Lagrangian function is given by

$$\begin{aligned} &\tilde{L}(\boldsymbol{\Theta}, \boldsymbol{\omega}, \lambda) \\ &\quad = -\sum_u \log(P(I_u, u|\boldsymbol{\Theta})) + \lambda \left(\sum_{k=0}^{K+1} \omega_k - 1 \right). \end{aligned} \tag{4}$$

At a local extremum, we have

$$\begin{aligned} 0 &= -\frac{\partial}{\partial \Theta_l} \tilde{L}(\boldsymbol{\Theta}, \boldsymbol{\omega}, \lambda) \\ &= \frac{\partial}{\partial \Theta_l} \sum_u \log \left(\sum_{k=0}^{K+1} \omega_k P(I_u, u|\Theta_k) \right) \\ &= \sum_u \frac{\omega_l (\partial / \partial \Theta_l) P(I_u, u|\Theta_l)}{\sum_{k=0}^{K+1} \omega_k P(I_u, u|\Theta_k)} \\ &= \sum_u p_{u,l} \frac{\partial}{\partial \Theta_l} \log(P(I_u, u|\Theta_l)), \end{aligned} \tag{5}$$

where $l = 1, \ldots, K$, and $p_{u,l}$ is the probability that pixel u stems from the lth process

$$p_{u,l} = \frac{\omega_l P(I_u, u|\Theta_l)}{\sum_{k=0}^{K+1} \omega_k P(I_u, u|\Theta_k)}. \tag{6}$$

Similarly, we obtain

$$0 = \frac{\partial}{\partial \lambda} \tilde{L}(\boldsymbol{\Theta}, \boldsymbol{\omega}, \lambda) = \sum_{k=0}^{K+1} \omega_k - 1. \tag{7}$$

$$0 = \frac{\partial}{\partial \omega_l} \tilde{L}(\Theta, \omega, \lambda)$$
$$= -\frac{P(I_u, u|\Theta_l)}{\sum_{k=0}^{K+1} \omega_k P(I_u, u|\Theta_k)} + \lambda$$
$$= -\frac{1}{\omega_l} \sum_u p_{u,l} + \lambda. \quad (8)$$

The parameters describing the observed environment should be calculated by solving Eqs. (5), (7) and (8). These equations can only be solved iteratively. A good iterative approach for this problem is provided by the EM algorithm, in which this is done by first calculating the probabilities $p_{u,l}$ using the current estimate for Θ and ω (the expectation step) and then solving Eqs. (5), (7) and (8) as if $p_{u,l}$ were constants independent of Θ (the maximization step). This process is repeated until convergence.

The expectation step consists of calculating the probabilities (6) and is theoretically trivial, although it takes most of the processing time in practice. In the following, we concentrate on the maximization step, i.e. the calculation of the unknown parameters given the probabilities $p_{u,l}$. λ, $\omega_0, \ldots, \omega_{K+1}$ can be calculated directly regardless of the choice of probability distribution for Θ. Taking into account that $\sum_{l=0}^{K+1} p_{u,l} = 1$ (see Eq. (6)), the solution to Eqs. (7) and (8) turns out to be $\lambda = N$, where N is equal to the number of pixels, and

$$\omega_l = \frac{1}{N} \sum_u p_{u,l}, \quad l = 0, \ldots, K+1. \quad (9)$$

Intuitively, ω_l is proportional to the percentage of pixels stemming from the process Θ_l.

To calculate the rest of the parameters, we must first decide how to model the process distributions Θ_k. Researchers have used various features when modeling images by mixture models such as, e.g., intensity variations [6], color [2,9], optical flow combined with the spatial coherence [3], and 3D ellipsoidal models [7]. The distribution of these features is usually modeled as Gaussian, which significantly simplifies the calculation of logarithms of probabilities in Eq. (5).

Our approach uses shape and color mixtures (or sometimes color intensity mixtures) to evaluate the probability that a pixel belongs to a certain process. Assuming that these properties are independent of each other, we can write

$$P(I_u, u|\Theta_l) \sim p(I_u|\Theta_l) p(u|\Theta_l). \quad (10)$$

In many cases, for example, when tracking body parts, the 2D shape of the tracked objects is roughly ellipsoidal and we can estimate it by the center of the object's image x_l and by the covariance matrix Σ_l of pixels contained in it. The shape part of the probability that a pixel u belongs to a blob can then be estimated as

$$p(u|\Theta_l) = \frac{1}{2\pi \sqrt{\det(\Sigma_l)}} \times \exp\left(-\tfrac{1}{2}(x - x_l)^T \Sigma_l^{-1} (x - x_l)\right). \quad (11)$$

Assuming that the object's texture consists of a finite number of colors, we can model the color probabilities by a Gaussian mixture model

$$p(I_u|\Theta_l) = \sum_{k=1}^{K_l} \omega_{l,k} p(I_u|\overline{I_{l,k}}, \Gamma_{l,k}), \quad (12)$$

where $\sum_{k=1}^{K_l} \omega_{l,k} = 1$ and

$$p(I_u|\overline{I_{l,k}}, \Gamma_{l,k}) = \frac{1}{\sqrt{(2\pi)^{2 \text{ or } 3} \det(\Sigma_l)}} \times \exp\left(-\tfrac{1}{2}(I_u - \overline{I_{l,k}}) \Gamma_{l,k}^{-1} (I_u - \overline{I_{l,k}})\right). \quad (13)$$

We experimented both with colors and color intensities, therefore 2 or 3 in Eq. (13) depending on the dimension of I_u.

The adaptation of colors $\overline{I_{l,k}}$ and their covariances $\Gamma_{l,k}$ within the EM algorithm makes the tracking unstable, therefore we keep them constant. The necessary parameters are determined in an off-line initialization phase. This means that

$$\frac{\partial}{\partial \Theta_l} \log(p(u|\Theta_l) p(I_u|\overline{I_{l,k}}, \Gamma_{l,k})) = \frac{\partial}{\partial \Theta_l} \log(p(u|\Theta_l)). \quad (14)$$

The parameters to be estimated are the objects' positions x_l and covariances $\Gamma_{l,k}$ and sometimes the color mixture probabilities $\omega_{l,k}$. Writing

$$p_{u,l,k} = \frac{\omega_{l,k} p(I_u|\overline{I_{l,k}}, \Gamma_{l,k})}{\sum_{k=1}^{K_l} \omega_{l,k} p(I_u|\overline{I_{l,k}}, \Gamma_{l,k})}, \quad (15)$$

we can transform Eq. (5) into

$$0 = \sum_u p_{u,l} \frac{\partial}{\partial \Theta_l} \log(\mathrm{P}(I_u, u|\Theta_l))$$

$$= \sum_u p_{u,l} \sum_{k=1}^{K} p_{u,l,k} \frac{\partial}{\partial \Theta_l} \log(p(u|\Theta_l) p(I_u|\overline{I_{l,k}}, \Gamma_{l,k}))$$

$$= \sum_u p_{u,l} \frac{\partial}{\partial \Theta_l} \log(p(u|\Theta_l)). \quad (16)$$

It is well known that these equations can be solved by computing the weighted mean and covariances of image pixels with $p_{u,l}$ being used as weights.

The reasoning when estimating the color mixture probabilities $\omega_{l,k}$ is similar as in the case of the estimation of blob mixture probabilities ω_l. The result is

$$\omega_{l,k} = \frac{1}{\sum_u p_{u,l}} \sum_u p_{u,l} p_{u,l,k}$$

$$= \frac{1}{N\omega_l} \sum_u p_{u,l} p_{u,l,k}, \quad (17)$$

where ω_l are the *newly* calculated blob mixture probabilities. This completes the basic algorithm used by our tracker.

2.2. Real-time considerations

The resolution of color images, which are captured by our system at 30 Hz, is 320 × 240. This means that we need to process ca. 6.6 MB of data every second. One way to reduce the computation time would be to model properties of the tracked entities by a simpler probability distribution than the normal distribution. This was done in [5], although in a somewhat different setting. We decided rather to stay with the normal distribution and to reduce the computation time by narrowing the areas in the image in which the probabilities need to be evaluated.

The regions of interest are first determined by windows located at the previous or predicted position of each tracked object. This works fine for compact objects whose minor and major axes do not differ too much. But a window is only a poor approximation for elongated objects such as sticks in Fig. 1. Therefore, we also generate an ellipsoidal mask around the tracked object. The mask is specified by a binary image having 1 at pixels where the probabilities need to be evaluated.

To further reduce the computation time, we process images at two different resolutions. First we run the EM algorithm on the reduced resolution image, i.e. 160 × 120. In our experiments, the window and the mask size are typically set to be 1.75 to 2 times larger than the minimal bounding box containing the object. The initial position is taken to be the previous or the predicted position, but the initial object size is set to be 1.5 to 2 times larger than the object in the previous image so that also rapidly moving objects can be tracked. After one or two iteration steps, we increase the resolution to the full resolution, but with the window and mask size reduced to only 1.25 of the initial object size. The initial position and shape in this second iteration are taken to be the position and shape estimated at the lower resolution. Again, only one or two steps of the EM algorithm are performed. For very

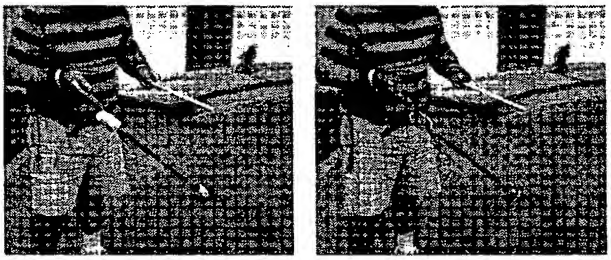

Fig. 1. Observing the devil sticking: (left) input; (right) result. Five blobs were tracked simultaneously.

big objects, we do not carry out the full resolution iteration at all because their position and shape can be determined reliably already in the lower resolution image and because the processing of big objects become very expensive at the full resolution.

We made use of the free Intel Image Processing Library, which is available from http://developer.intel.com/software/products/perflib/, to implement our system. The library is optimized for various Intel Pentium processors and is effective at taking advantage of the MMX technology. It also includes support for windowing and masking, thus making it suitable for the development of real-time vision systems like ours.

2.3. Shape estimation

The size (the major and the minor axes a and b) and the orientation (θ) of the blob $\mathcal{B}$ can be calculated using the relationship between these parameters and the covariance of the pixels within the blob. In the ideal case, we can make a crisp decision whether a pixel belongs to a blob or not. The covariance of the pixels is then given by

$$\begin{aligned}
&\frac{1}{\pi ab}\int_{\mathcal{B}}\begin{bmatrix}(x-x_0)^2 & (x-x_0)(y-y_0)\\ (x-x_0)(y-y_0) & (y-y_0)^2\end{bmatrix}\mathrm{d}x\,\mathrm{d}y\\
&\quad=\frac{1}{\pi ab}\boldsymbol{R}^{\mathrm{T}}\left(\int_{x^2/a^2+y^2/b^2\leqslant 1}\begin{bmatrix}x^2 & xy\\ xy & y^2\end{bmatrix}\mathrm{d}x\,\mathrm{d}y\right)\boldsymbol{R}\\
&\quad=\boldsymbol{R}^{\mathrm{T}}\begin{bmatrix}\frac{1}{4}a^2 & 0\\ 0 & \frac{1}{4}b^2\end{bmatrix}\boldsymbol{R}, \qquad (18)
\end{aligned}$$

where (x_0, y_0) is the center of the ellipse and $\boldsymbol{R}$ the rotation matrix aligning the ellipse with the coordinate axes. It follows that we can estimate the size and the orientation of the tracked object by solving the eigenvalue problem for the estimated covariance matrix. The lengths of the major and the minor axes are given by $a = 2\sqrt{\lambda_1}$ and $b = 2\sqrt{\lambda_2}$, where λ_1 and λ_2 are the larger and the smaller eigenvalue of the estimated covariance matrix, respectively. The rotation matrix $\boldsymbol{R}$ (and from it the angle θ) is given by a matrix with the corresponding eigenvectors in its columns. These parameters can be used to generate masks for efficient processing and to draw results into the captured images. This is how the result images shown in this paper were generated.

2.4. Occlusion reasoning

Occlusions are common in real environments, especially when observing humans manipulating objects, such as in Fig. 1. Occlusion reasoning usually requires a prediction step, as for example, in [8], on the basis of which we can determine which objects will overlap in the next image frame and where.

In general, we have no information about the physics of motion of the observed objects, therefore we predict each parameter using a discrete second-order dynamical system

$$x(t) = ax(t-1) + bx(t-2) + e(t), \qquad (19)$$

where $x(t)$ is one of the parameters describing the tracked object (position, orientation, shape) and $e(t)$ is the system noise, both given at time t. The unknown parameters a and b are estimated using recursive least-squares with a forgetting factor.

Once the object positions in the next frame are estimated, we can find the regions of occlusion. The decision which of the two overlapping objects is in front can be made using the estimated 3D positions (see Section 2.5) or prior knowledge. If we predict that at a certain pixel one object will overlap the other and if in the next image frame we really find evidence that the pixel was generated by the first object, then we assign the estimated probability of the first object's appearance (10) not only to the object itself, but also to the overlapped object. Since we have less confidence that the second object really projects onto this pixel, we assign only half of the actually estimated probability to the second object. If the probability that the second object projects onto this pixel is greater than the probability of the supposedly overlapping object, then this probability is retained.

We can reason about occlusions only for objects that are actually being tracked. We cannot say much when one of the tracked objects is occluded by something that our system cannot perceive. This would require more complicated shape analysis that we want to avoid. Our approach is based on the prediction of future positions and can thus work only when the predictions are reliable. We have obtained good results when using a high-speed camera as in Fig. 1. However, the approach becomes less reliable when tracking rapidly moving objects with standard video cameras as in our real-time system.

2.5. 3D position estimation

We use stereo to estimate the position of tracked entities such as hand or head. We could take the center of blobs in both images to estimate the 3D position of a tracked body part. However, the two centers give only a rather crude stereo correspondence because the blobs found in the two images do not cover the same areas on the body (see Fig. 2). This happens because of differences in the viewing direction and because of uncertainties in the estimation of shape.

Cross-correlation is a standard method for the calculation of stereo correspondences. In our case, we are not interested in generating a full depth map, but only to estimate a 3D blob position. We take the center of the blob in the left image as a starting point. A box template around the blob center is extracted and we attempt to find the best match in the right image using zero mean normalized cross-correlation

$$\mathrm{ZNCC}_{l,r}(u, u+d) = \frac{\mathrm{cov}(I_{u,l}, I_{u+d,r})}{\sqrt{\mathrm{var}(I_{u,l})}\sqrt{\mathrm{var}(I_{u+d,r})}}, \tag{20}$$

where

$$\mathrm{cov}(I_{u,l}, I_{u+d,r}) = \frac{\sum_{\Delta\in\mathcal{T}}(I_{u+\Delta,l} - \overline{I_{u,l}})^{\mathrm{T}}(I_{u+d+\Delta,r} - \overline{I_{u+d,r}})}{(n-1)},$$

$$\mathrm{var}(I_u) = \frac{\sum_{\Delta\in\mathcal{T}}\|I_{u+\Delta} - \overline{I_u}\|^2}{(n-1)}.$$

and $\overline{I_u}$ is the mean color within the box around pixel u. The maximum of correlation (20) is sought for in a region defined by a slice of the image along the epipolar line that lies within the right blob.

3. Experiment 1: Mimicking of hand motion

The first example on which we show the usefulness of the developed vision system is mimicking of human hand motion. The task of the humanoid is to move its palm along the same type of path as the human demonstrator. Fig. 3 shows an example of mimicking a circular motion.

DB first records the initial demonstrator's hand position as detected by his vision system. At the same time, DB's hand position is also recorded. As the demonstrator starts moving his hand, DB determines the demonstrator's hand motion relative to the initial hand position and generates points in his workspace that result in the same path relative to the initial DB's hand position. These 3D points are transmitted to DB's control system in real time as the desired hand positions. The Cartesian hand positions are transformed, again in real time, into DB's joint angles using an inverse kinematics method described in [13]. The resulting joint angle positions are followed by DB's controller.

The main problem with this approach is that 3D point positions are very noisy both because of the uncertainties in our vision system and because of vibrations caused by DB's motion. To alleviate this problem, we developed a method in which DB does

Fig. 2. The detected right human hand overlayed by the estimated blob as seen by left and right eye.

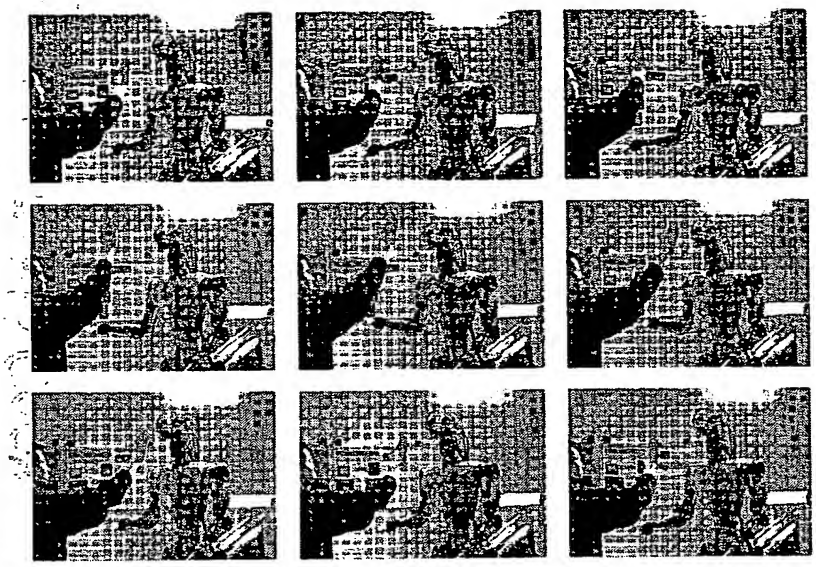

Fig. 3. DB mimicking hand motion. Every tenth frame from a video is shown.

not start moving along the demonstrated path immediately but first gathers 1 second of data (30 points on the demonstrated path). A smooth trajectory is then generated using least-squares approximation with B-splines. The applied B-spline basis should have significantly less basis functions than there are data points. In the example in Fig. 4, we used only six basis functions for 1 second of data. In the next second, DB starts moving its hand along the generated spline trajectory while continuing to monitor the demonstrator's motion. This process is repeated every second and continuity constraints are enforced at the edges. Such an approach seems to be natural because also humans first try to figure out what is going on before they start mimicking other people's motion.

Another important issue is the choice of arm configuration. A humanoid arm is redundant with respect to the mimicking of hand motion and there is an infinite number of configurations that result in the same hand motion. In the experiment in Fig. 3, the robot arm configuration was different from the demonstrator's arm configuration when the mimicking began and this difference was retained throughout the mimicking session. Ideally, the visual system should recognize the initial arm configuration, but this is a formidable task for visual processing, especially if it is to be performed in real time. We are currently working on this problem.

Mimicking arm motion with a humanoid has been done before. The approach in [4] is also based on the tracking of hand motion, but the authors map the 3D Cartesian motion into the joint space using some predefined rules and without using any kinematic information. While this can be effective in some cases, it is rather arbitrary and cannot account for all possible motions. In addition, their vision system

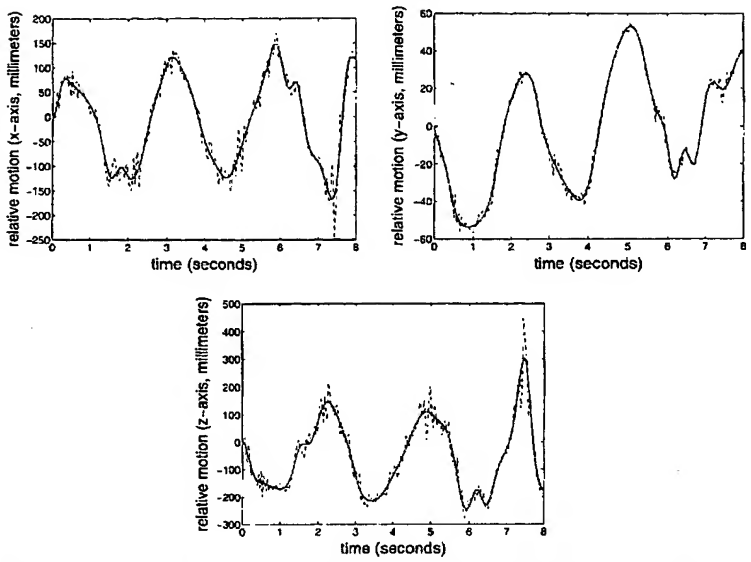

Fig. 4. Least-squares approximation of the generated trajectory with B-splines: (solid line) the smoothed trajectories; (dashed line) the original trajectories.

is based on a more standard processing of visual information.

4. Experiment 2: Smooth pursuit

Very accurate oculomotor control is required in order to track a moving target in the viewfield of the narrow angle foveal cameras. Primates can precisely follow a target moving, e.g., at constant velocity or in sinusoidal motion despite significant processing delays in the visual pathway. Furthermore, their oculomotor control sometimes becomes predictive. This cannot be achieved by simple visual feedback control.

In our system, smooth pursuit oculomotor control is achieved by a biologically inspired and control theoretically sound controller consisting of two cascaded learning modules. The goal of the first one is to learn an inverse model of the oculomotor system, while the other tries to learn the dynamics of a visual target to predict the current target velocity in the head coordinates. Although both modules work together to minimize the tracking error in the image, smooth pursuit is usually conducted with the assumption that the inverse model was learned beforehand in order to guarantee that the predictor can learn the proper visual target dynamics. We employ LWPR (locally weighted projection regression) for the integrated learning of both modules [16]. It enables on-line learning of higher-order linear/nonlinear dynamics.

Fig. 5 shows the operation of our vision system combined with the described on-line learning algorithm. The accuracy of the approach is demonstrated in Fig. 6. The final rectified mean error reached less

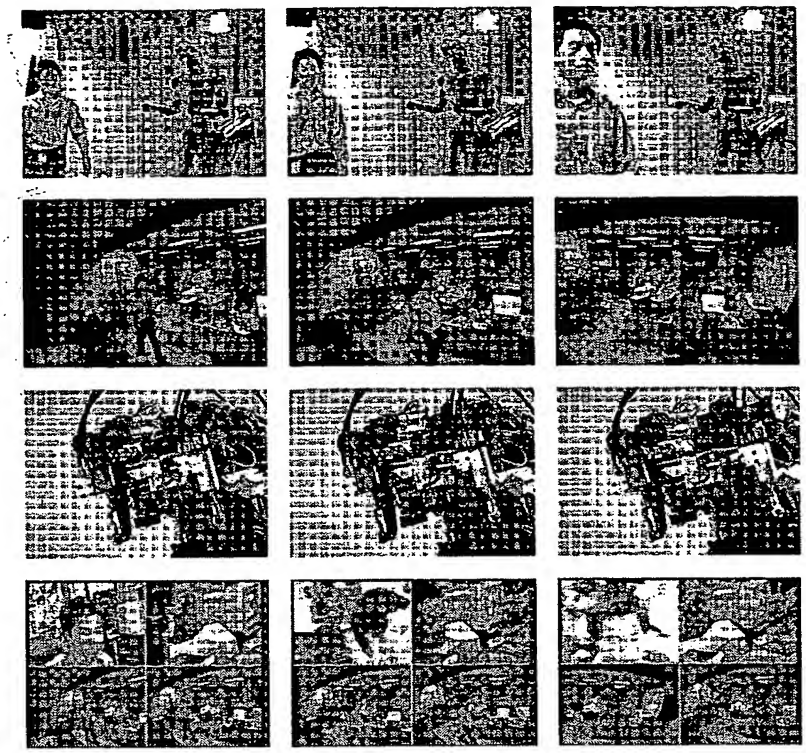

Fig. 5. Smooth pursuit with DB's left eye. First row shows the action taken by the external camera, second row shows successful face tracking while the camera is moving (the face is overlayed with the detected blob), third row shows how DB's left eye is moving, and fourth row shows the view from all four cameras demonstrating that the person's head stays within the viewfield of the left narrow-view camera.

than 0.05 rad, which is very small even for our foveal vision. As expected, learning the dynamics of head motion was more difficult than learning the dynamics of an ideal pendulum that we used in our initial experiments, but the learning algorithm was nevertheless successful. Because of the initial values of learning parameters, it did happen occasionally that the learning algorithm produced wrong models, which resulted in wild eye motions and temporary loss of the visual target. It turned out that our vision system is reliable enough to recover from such errors so that the learning system was able to start learning again after failure. A more detailed description of this algorithm can be found in [12].

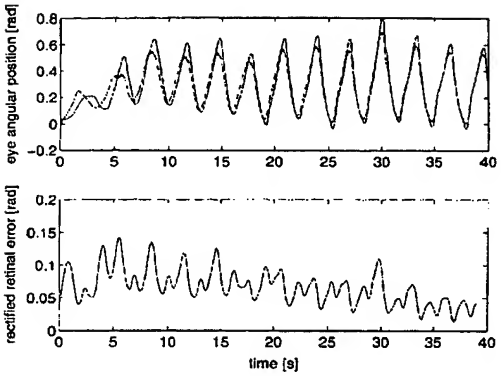

Fig. 6. Top: time course of the estimated target position angle (dotted line) and the eye angular position (solid line). Bottom: time course of the rectified mean retinal error.

5. Summary and conclusion

We have presented a real-time visual system that enables a humanoid robot to interact with humans. Viewing the motion tracking and estimation problem in a probabilistic setting allowed us to avoid the excessive simplicity of real-time systems based on some form of thresholding and ensured that the system is not too brittle with respect to the setting of initial parameters. In fact, there are only a few parameters that need to be set in our system. In addition, the developed algorithm is simple enough so that a real-time implementation was possible. Compared to other probabilistic real-time systems, most notably among them the "camshift" algorithm which is included in the publicly available OpenCV library [10], our system considers not only the distribution of color but also the spatial distribution of pixels. This becomes especially important when tracking multiple objects and when it is necessary to reason about occlusions.

We performed numerous experiments with the proposed approach and used it for the generation of several simple behaviors such as smooth pursuit and on-line mimicking of human hand motion. Overall, the system proved to work reliably when used in complex environments such as the ones in Figs. 1–3 and 5, and to be reasonably insensitive to moderate variations in the lighting conditions.

Acknowledgements

Most of this work was done at ATR, Kyoto, Japan, in the frame of CyberHuman Project and ERATO Kawato Dynamic Brain Project. Support for Chris Atkeson was also provided by US National Science Foundation Award IIS-9711770. Support for Aleš Ude was also provided by the Slovenian Ministry of Education, Science and Sport.

References

[1] C.G. Atkeson, J. Hale, F. Pollick, M. Riley, S. Kotosaka, S. Schaal, T. Shibata, G. Tevatia, A. Ude, S. Vijayakumar, M. Kawato, Using humanoid robots to study human behavior, IEEE Intelligent Systems 15 (4) (2000) 46–56.

[2] G.R. Bradski, Computer vision face tracking for use in a perceptual user interface, Intel Technology Journal Q2 (1998), http://developer.intel.com/technology/itj.

[3] C. Bregler, Learning and recognizing human dynamics in video sequences, in: Proceedings of the IEEE Computer

Society Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico, June 1997.
[4] G. Cheng, Y. Kuniyoshi, Real-time mimicking of human body motion by a humanoid robot, in: Proceedings of the Sixth International Conference on Intelligent Autonomous Systems (IAS2000), Venice, Italy, July 2000, pp. 273–280.
[5] D. Comaniciu, V. Ramesh, P. Meer, Real-time tracking of non-rigid objects using mean shift, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Hilton Head, SC, June 2000, Vol. 2, pp. 142–149.
[6] A. Jepson, M. Black, Mixture models for image representation, Technical Report ARK96-PUB-54, PRECARN ARK Project, Department of Computer Science, University of Toronto, Toronto, Ont., March 1996.
[7] N. Jojic, M. Turk, T.S. Huang, Tracking self-occluding articulated objects in dense disparity maps, in: Proceedings of the IEEE International Conference on Computer Vision, Kerkyra, Greece, 1999, pp. 123–130.
[8] D. Koller, J. Weber, J. Malik, Robust multiple car tracking with occlusion reasoning, in: Proceedings of the Third European Conference on Computer Vision (Computer Vision — ECCV'94), Stockholm, Sweden, 1994, pp. 189–196.
[9] S.J. McKenna, Y. Raja, S. Gong, Tracking colour objects using adaptive mixture models, Image and Vision Computing 17 (3–4) (1999) 225–231.
[10] Open Source Computer Vision Library (OpenCV). http://www.intel.com/research/mrl/research/opencv/.
[11] S. Schaal, Is imitation learning the route to humanoid robots? Trends in Cognitive Sciences 3 (6) (1999) 233–242.
[12] T. Shibata, S. Schaal, Biomimetic smooth pursuit based on fast learning of the target dynamics, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Maui, HI, 2001.
[13] G. Tevatia, S. Schaal, Inverse kinematics for humanoid robots, in: Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000, pp. 294–299.
[14] A. Ude, Robust estimation of human body kinematics from video, in: Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems, Kyongju, Korea, October 1999, pp. 1489–1494.
[15] A. Ude, C.G. Atkeson, M. Riley, Planning of joint trajectories for humanoid robots using B-spline wavelets, in: Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000, pp. 2223–2228.
[16] S. Vijayakumar, S. Schaal, Fast and efficient incremental learning for high-dimensional movement systems, in: Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, April 2000, pp. 1894–1899.

Aleš Ude studied Applied Mathematics at the University of Ljubljana, Slovenia, and Computer Science at the University of Karlsruhe, Germany, where he received a doctoral degree in 1996. From 1998 to 2000, he was an STA Fellow in the Kawato Dynamic Brain Project, ERATO, JST. Currently, he holds a research position at the Jožef Stefan Institute, Ljubljana, Slovenia, and is also associated with the CyberHuman project, ATR-I, Kyoto, Japan. His research interests include the visual perception of human activity and its application to the robot learning.

Tomohiro Shibata received his Ph.D. in Information Engineering from the University of Tokyo in 1996. From 1996 to 1997, he was a Postdoctoral Fellow at the University of Tokyo. Currently, he is a researcher in the Japan Science and Technology Corporation. He has been a member of the ERATO project led by Dr. Mitsuo Kawato since April 1997. His hope is to contribute to the brain science based on the knowledge of robotics. He works on modeling and learning of biologically plausible oculomotor controllers.

Christopher G. Atkeson is an Associate Professor at the Robotics Institute and Human–Computer Interaction Institute at Carnegie Mellon University, Pittsburgh, Pennsylvania. His research focuses on numerical approaches to machine learning, and uses robotics and intelligent environments as domains in which to explore the behavior of learning algorithms. He is a recipient of a National Science Foundation Presidential Young Investigator award.